Une Plate-forme d'Emulation Légère pour Etudier les Systèmes Pair-à-Pair

Lucas Nussbaum, Olivier Richard

Une Plate-forme d'Emulation Légère pour Etudier les Systèmes Pair-à-Pair

RECHERCHE/SYNTHESE

Lucas Nussbaum — Olivier Richard

Laboratoire Informatique et Distribution - IMAG / Projet INRIA Mescal
ENSIMAG - Antenne de Montbonnot - ZIRST
51 avenue Jean Kuntzmann, 38330 Montbonnot Saint-Martin, France
{Lucas.Nussbaum,Olivier.Richard}@imag.fr

RÉSUMÉ. *Les méthodes actuellement les plus utilisées pour étudier les systèmes pair-à-pair (modélisation, simulation, et exécution sur des systèmes réels) montrent souvent des limites sur les plans du passage à l'échelle et du réalisme. Cet article présente P2PLab, une plate-forme pour l'étude et l'évaluation des systèmes pair-à-pair, qui combine l'émulation (utilisation de l'application réelle à étudier à l'intérieur d'un environnement synthétique) et la virtualisation. Après la présentation des caractéristiques principales de P2PLab (émulation réseau distribuée, virtualisation légère), nous montrons son utilité lors de l'étude du système de diffusion de fichiers BitTorrent, notamment en comparant deux implantations différentes de ce protocole complexe.*

ABSTRACT. *The current methods used to study and evaluate peer-to-peer systems (namely modelisation, simulation, and execution and real-world systems) often show limits regarding scalability and accuracy. This paper describes P2PLab, an evaluation platform for peer-to-peer systems, which combines emulation (execution of the real application inside a synthetic environment) and virtualization. After describing the main features of P2PLab (distributed network emulation, lightweight virtualization), we demonstrate its usefulness by comparing two implementations of the BitTorrent file-sharing system.*

MOTS-CLÉS : *systèmes pair-à-pair, évaluation, émulation, virtualisation, BitTorrent*

KEYWORDS: *peer-to-peer systems, evaluation, emulation, virtualization, BitTorrent*

RSTI - TSI – 27/2008. RenPar'06, pages 1 à 22

1. Introduction

Les systèmes pair-à-pair et les algorithmes utilisés par ces systèmes ont fait l'objet d'un intérêt considérable ces dernières années. Ils sont devenus plus performants, mais aussi plus complexes, ce qui a eu pour conséquence de les rendre plus difficiles à concevoir, à vérifier et à évaluer. Il devient nécessaire de pouvoir vérifier qu'une application se comportera correctement même lorsqu'elle sera exécutée sur des milliers de nœuds, ou de pouvoir comprendre des applications conçues pour fonctionner sur un tel nombre de nœuds.

Les applications distribuées sont traditionnellement étudiées en utilisant la modélisation, la simulation, et l'exécution sur des systèmes réels. La modélisation et la simulation consistent à utiliser un modèle du fonctionnement de l'application dans un environnement synthétique (par opposition à un environnement réel). Cette méthode est largement utilisée, et permet d'obtenir des résultats de bonne qualité assez facilement. Toutefois, il est souvent nécessaire d'accepter un compromis entre le nombre de nœuds simulés et la précision du modèle : il est difficile de simuler efficacement un grand nombre de nœuds en utilisant un modèle complexe de l'application.

L'alternative consiste à exécuter l'application réelle à étudier dans un environnement d'expérimentation réel comme PlanetLab (Chun *et al.*, 2003). Mais ces plates-formes sont difficiles à contrôler et à modifier (afin de les adapter à des conditions d'expériences particulières), et les résultats sont souvent difficiles à reproduire, car les conditions environnementales peuvent varier de manière importante entre les expériences. L'expérimentation sur ce type de plate-forme est nécessaire lors du développement d'un système pair-à-pair, mais ces expériences ne sont pas suffisantes, car il est très difficile de couvrir ainsi suffisamment de conditions d'expériences différentes pour pouvoir conclure dans le cas général. L'utilisation d'autres méthodes d'évaluation devient alors de plus en plus importante (Haeberlen *et al.*, 2006).

Entre ces deux approches, il est nécessaire d'explorer d'autres voies, permettant d'exécuter l'application réelle dans un environnement synthétique, sous des conditions reproductibles. Cet article décrit ainsi une solution intermédiaire, combinant émulation et virtualisation, et montre que cette approche permet d'obtenir des résultats intéressants lors de l'étude d'applications existantes.

2. Emulation et virtualisation

Il est nécessaire de commencer par distinguer émulation et virtualisation.

L'émulation consiste à exécuter l'application à étudier dans un environnement modifié afin de correspondre aux conditions expérimentales souhaitées. Il est nécessaire de déterminer quelles sont les ressources à émuler (et avec quelle précision) : il s'agit souvent d'un compromis entre réalisme et coût. Par exemple, lors de l'étude de systèmes pair-à-pair, l'émulation réseau est importante, alors que l'émulation précise des entrées/sorties sur le disque dur n'est probablement

pas nécessaire. L'émulation est souvent coûteuse (temps processeur, mémoire), et son coût est souvent difficile à évaluer, car il dépend à la fois de la qualité de l'émulation et de ses paramètres : émuler un réseau à latence importante nécessitera une quantité de mémoire plus importante qu'émuler un réseau à faible latence, à cause de la nécessité de conserver les paquets dans une file d'attente (par exemple, émuler une latence de 1s sur un réseau avec un débit de 1 Gbps requiert une file d'attente occupant 125 Mo).

La virtualisation de ressources permet de partager une ressource réelle entre plusieurs instances d'une application. Elle est nécessaire afin de permettre l'étude d'un grand nombre de nœuds. Dans le cas des systèmes distribués, la virtualisation permet d'exécuter plusieurs instances de l'application ou du système d'exploitation sur chaque machine physique. Bien entendu, comme les ressources réelles de la machine physique sont partagées entre les différentes instances, l'équité est un problème important. Comme la précision de l'émulation, le niveau d'équité de la virtualisation est un compromis entre qualité et coût : un bon niveau d'équité s'obtient souvent au détriment des performances globales de la machine physique.

3. Etat de l'art

Beaucoup de travaux ont été effectués récemment sur la virtualisation, avec différentes approches. Linux Vserver (Project., 2003) est une modification du noyau Linux qui y ajoute des *contextes* et contrôle les interactions entre ces contextes, permettant à plusieurs environnements de partager le même noyau de manière invisible pour les applications, avec un très faible surcoût. User Mode Linux (Dike, 2000) est une adaptation du noyau Linux vers un processus Linux. Enfin, Xen (Barham *et al.*, 2003) utilise la *para-virtualisation* (exécution de systèmes légèrement modifiés au-dessus d'un système hôte) pour permettre d'exécuter simultanément plusieurs systèmes d'exploitation.

Le réseau est l'aspect le plus important à contrôler lorsqu'on s'intéresse aux systèmes pair-à-pair, puisqu'il s'agit du principal facteur limitant. Des outils de bas niveau permettent d'émuler différentes caractéristiques de liens (en faisant varier la bande passante et la latence), tandis que des outils de haut niveau permettent de construire des topologies synthétiques complexes.

Dummynet (Rizzo, 1998), sous FreeBSD, est l'émulateur réseau le plus utilisé. Il est intégré au pare-feu de FreeBSD, et est donc contrôlé par l'ajout de règles spécifiques. NISTNet (Carson *et al.*, 2003) (sous Linux 2.4) et Linux Traffic Control (TC) (Hemminger, 2005) sous Linux 2.6 ont des fonctionnalités similaires. Ces outils se positionnent au niveau paquet et ordonnancent les paquets entrants et/ou sortants du système pour contrôler le débit et le délai, en émulant aussi des problèmes comme la perte de paquets.

Des outils de plus haut niveau permettent de recréer des topologies virtuelles : NetBed (White *et al.*, 2002) est une plate-forme combinant des nœuds réels (utilisant des connexions RTC ou ADSL), des nœuds utilisant Dummynet, et des nœuds simulés (en utilisant Network Simulator Emulation Layer) pour fournir un environnement expérimental. Mais en raison de son faible nombre de nœuds (environ 200), il est assez difficile d'y faire des expériences à très grande échelle. Modelnet (Vahdat *et al.*, 2002) utilise des nœuds d'une grappe partitionnée en deux sous-ensembles : l'application à étudier est exécutée sur les *edge nodes* tandis que les *Modelnet core nodes* émulent une topologie réseau. Modelnet utilise une phase de *distillation* pour faire un compromis entre réalisme et passage à l'échelle, permettant ainsi d'émuler le réseau sur un faible nombre de nœuds. Mais la mise en œuvre de Modelnet est difficile.

Les outils existants de virtualisation ciblent un grand réalisme et ont un facteur de virtualisation (nombre de machines virtuelles/nombre de machines physiques) assez faible. Ils virtualisent un système d'exploitation complet (noyau, bibliothèques, applications) alors que cela n'est souvent pas nécessaire dans le cadre de l'étude des systèmes pair-à-pair, qui sont généralement des applications *verticales*, ayant peu de dépendances avec le reste du système.

Un autre problème est l'émulation de la topologie du réseau : les outils existants visent une émulation réaliste du cœur du réseau (congestion, routage ...). Or la plupart des applications pair-à-pair sont utilisées par des personnes individuelles sur des liens de type ADSL. Certains aspects du cœur du réseau sont importants (la latence, notamment, pour des expériences impliquant des problèmes de localité), mais dans le cadre d'une plate-forme d'émulation, l'accent peut être mis sur l'émulation du lien entre les nœuds et leur fournisseur d'accès à internet. Ce lien est en effet le goulot d'étranglement dans la plupart des cas.

4. P2PLab

4.1. *Présentation générale*

P2PLab est notre outil d'étude des systèmes pair-à-pair utilisant l'émulation. Il vise une grande efficacité (un grand nombre de nœuds doivent pouvoir être étudiés sur un faible nombre de nœuds physiques) et un bon passage à l'échelle (des expériences avec plusieurs milliers de nœuds doivent être possibles).

P2PLab virtualise au niveau des processus, pas au niveau du système d'exploitation comme les autres outils de virtualisation le font. En effet, une virtualisation complète du système n'est pas forcément nécessaire lorsque les objets étudiés sont des applications pair-à-pair, ces applications n'ayant en général que peu de dépendances sur le reste du système. P2PLab utilise FreeBSD pour pouvoir utiliser Dummynet pour l'émulation réseau. Une approche décentralisée est utilisée pour émuler les topologies réseau, permettant un meilleur passage à l'échelle.

Dans la section suivante, nous présenterons le système de virtualisation proposé par P2PLab, après avoir vérifié que les caractéristiques de FreeBSD correspondaient bien à celles attendues. Puis nous nous intéresserons à son modèle d'émulation réseau.

4.2. *Virtualisation*

FreeBSD a été choisi pour P2PLab en raison de la disponibilité de Dummynet (Rizzo, 1997), l'émulateur réseau inclu dans FreeBSD. Mais il était nécessaire de commencer par vérifier que les caractéristiques de FreeBSD étaient suffisantes pour permettre l'exécution d'un grand nombre de processus sans fausser les résultats des expériences. FreeBSD 6 propose deux ordonnanceurs de processus différents :

– un ordonnanceur *historique* (appelé 4BSD) dérivé de celui de BSD 4.3 ;

– un ordonnanceur plus moderne, ULE (Roberson, 2003)[1], développé pour permettre une meilleure interactivité, même sous une charge importante.

Comme le nombre de processus concurrents sera important, l'ordonnanceur jouera un rôle-clé, et il est également important de comparer ces deux ordonnanceurs afin de sélectionner le plus adapté à l'exécution de nombreux processus concurrents.

4.2.1. *Adéquation de FreeBSD*

Dans une première expérience, nous avons démarré un grand nombre de processus utilisant intensivement le processeur en même temps et examiné le temps nécessaire à la complétion de l'ensemble des tâches. Cette évaluation, ainsi que toutes les suivantes, a été réalisée sur la plate-forme GridExplorer, qui fait partie du projet Grid'5000 (Cappello *et al.*, 2005). Les machines utilisées sont des Bi-Opteron 2 Ghz avec 2 Go de mémoire vive et un réseau Gigabit Ethernet. Le programme utilisé réalisait un calcul utilisant peu de mémoire (calcul de la fonction de Ackermann) et prenant de l'ordre de $3,3$ secondes. Afin d'éviter des disparités causés par les versions des compilateurs, le même code assembleur a été utilisé sur les différents systèmes. Comme les machines utilisées disposent de deux processeurs, les calculs se répartissent naturellement sur les deux processeurs, et le temps moyen par tâche est donc de l'ordre de 1,65 secondes.

La figure 1 montre qu'on n'observe pas de surcoût lié au nombre de processus concurrents. Au contraire, le temps moyen d'exécution semble même baisser très légèrement lorsque le nombre de processus augmente (surtout sous Linux 2.6), probablement à cause d'effets de cache ou d'économies d'échelle (certaines parties du coût n'étant pas dépendantes du nombre de processus, leur part diminue lorsque le nombre de processus augmente).

Nous avons ensuite réalisé la même expérience, mais avec des processus utilisant la mémoire de manière intensive (réalisant des opérations simples sur des matrices de

1. Il s'agit d'un jeu de mots, l'option pour l'activer étant appelée SCHED_ULE.

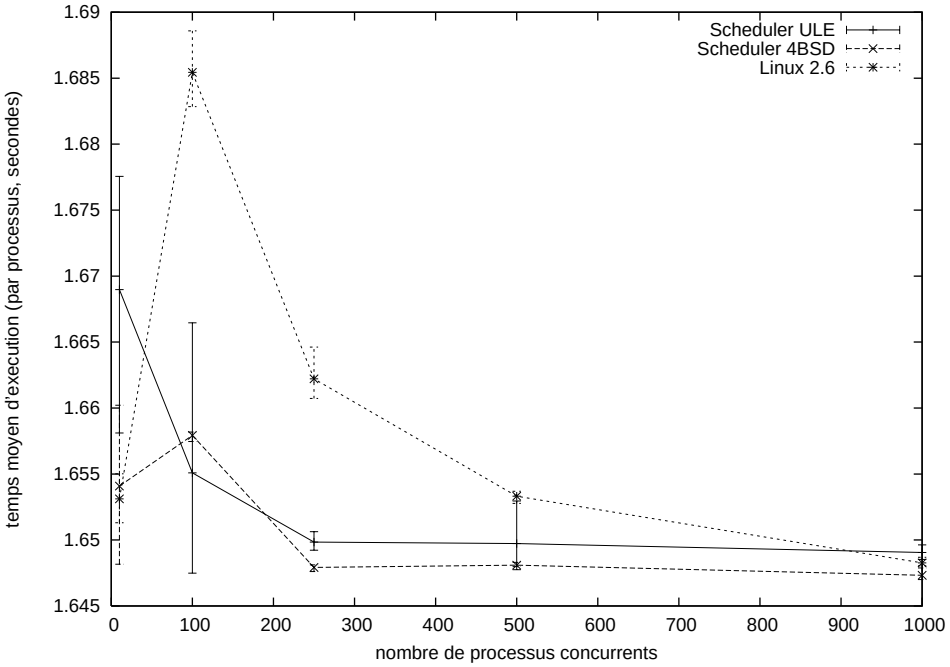

Figure 1. *Temps moyen d'exécution en fonction du nombre de processus concurrents. Les processus utilisent intensivement le processeur, mais n'utilisent que peu de mémoire*

grande taille). Exécutés seuls, ils ont besoin d'environ 3 secondes de temps processeur avant de terminer. La figure 2 permet d'abord de constater que les résultats obtenus avec Linux et FreeBSD sont très différents. Sous Linux 2.6, l'ordonnanceur et/ou la gestion mémoire permettent d'éviter au temps d'exécution d'augmenter lorsque l'ensemble des instances ne peuvent plus être contenues dans la mémoire vive. Au contraire, sous FreeBSD, dès que la mémoire virtuelle (*swap*) est utilisée, le temps d'exécution augmente de manière importante.

Dans les expériences futures, il sera important de se placer dans des conditions où l'utilisation de *swap* n'est pas nécessaire. Nous avons donc fait le choix de ne pas activer le swap sur les machines utilisées. Ainsi, un manque de mémoire se traduira par une erreur franche, facilement détectable, plutôt que par une perte progressive de performances.

L'équité entre les processus est également importante : certaines instances ne doivent pas bénéficier plus souvent ou plus longtemps du processeur. L'expérience suivante nous permet d'avoir une première estimation du niveau d'équité fourni par

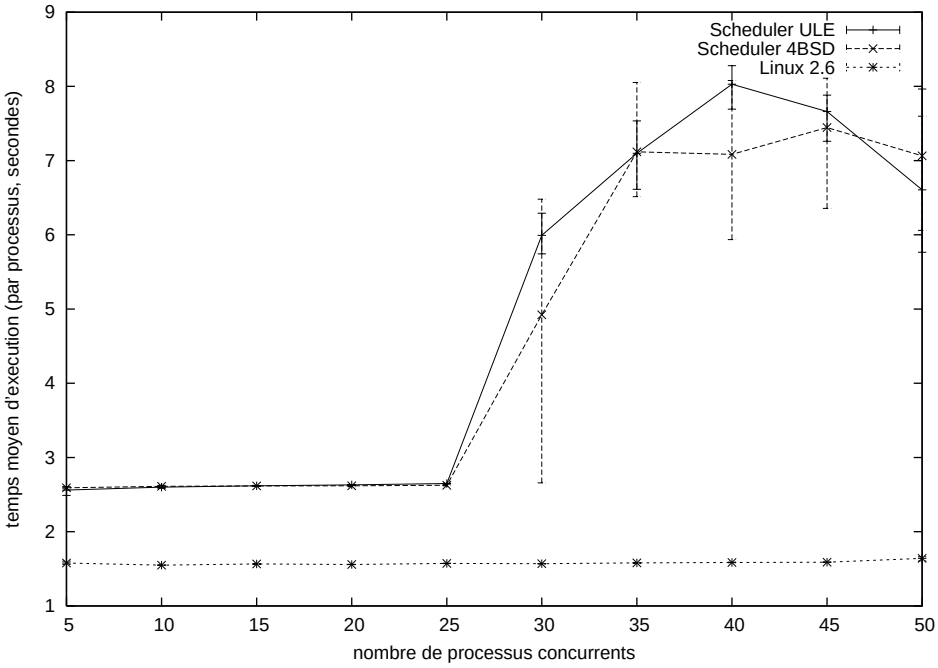

Figure 2. *Temps moyen d'exécution en fonction du nombre de processus concurrents. Les processus utilisent intensivement le processeur et la mémoire*

Linux et FreeBSD. Nous démarrons en « même temps »[2], sur la même machine, 100 instances d'un même programme utilisant intensivement le processeur, et mesurons les temps de terminaison de chaque instance. Le programme exécuté seul prend environ 5 secondes à terminer.

La figure 3 montre qu'avec l'ordonnanceur 4BSD et celui de Linux, la plupart des processus terminent *presque* en même temps. L'ordonnanceur ULE laisse par contre apparaître de plus grandes variations. Il faut noter que ces résultats sont différents de ceux qu'une précédente étude réalisée avec FreeBSD 5 (Nussbaum *et al.*, 2006) avait donnés : avec l'ordonnanceur ULE, certains processus étaient excessivement privilégiés et s'exécutaient seuls sur l'un des processeurs. Ce problème semble avoir été corrigé dans FreeBSD 6.

Même si cette approche fournit des résultats satisfaisants, tant sur le plan du passage à l'échelle que de l'équité, elle est limitée : elle ne permet pas, par exemple, de réaliser des expériences où des processeurs virtuels de vitesses différentes sont affec-

2. Un processus, lancé avec une priorité élevée, lance les instances qui sont exécutées à une priorité plus faible. Les résultats ne montrent pas une influence significative de l'ordre de lancement.

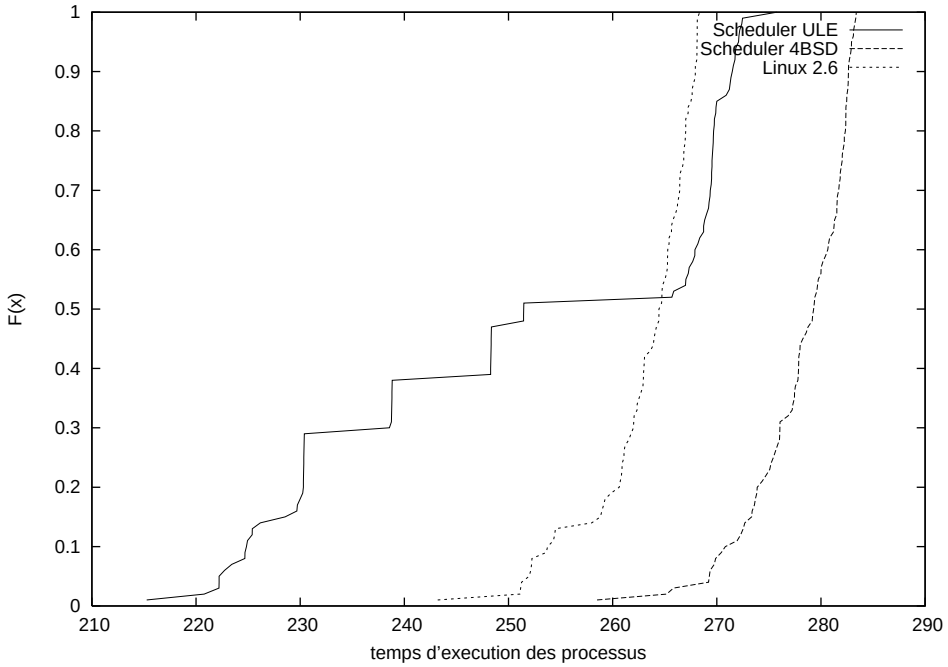

Figure 3. *Fonction de répartition des temps d'exécution des processus avec Linux 2.6 et les deux ordonnanceurs de FreeBSD. $F(x)$ indique la proportion des processus ayant terminé avant t*

tés aux processus. Cette approche n'est donc pas adaptée à l'étude des systèmes de type *Desktop Computing*. L'utilisation de solutions de virtualisation plus lourdes permettrait un contrôle plus précis du partage du processeur, et permettrait également de contrôler la mémoire.

Dans la suite des expériences, nous avons choisi d'utiliser l'ordonnanceur 4BSD, malgré cette limitation : il n'existe pas de solution sans cette limitation à l'heure actuelle sous FreeBSD.

4.2.2. *Virtualisation au niveau processus*

Comme indiqué précédemment, P2PLab virtualise au niveau de l'identité réseau des processus : les instances exécutées sur la même machine physique partagent toutes les ressources (système de fichiers, mémoire, etc.) comme des processus normaux, mais chaque processus virtualisé a sa propre adresse IP sur le réseau. L'adresse IP de chaque machine physique est conservée à des fins d'administration, tandis que les adresses IPs des nœuds virtuels sont configurées comme des *alias d'interface*, comme montré sur la figure 4 (la plupart des systèmes Unix, dont Linux et FreeBSD, permettent d'affecter plusieurs adresses IP à une même interface réseau à travers un

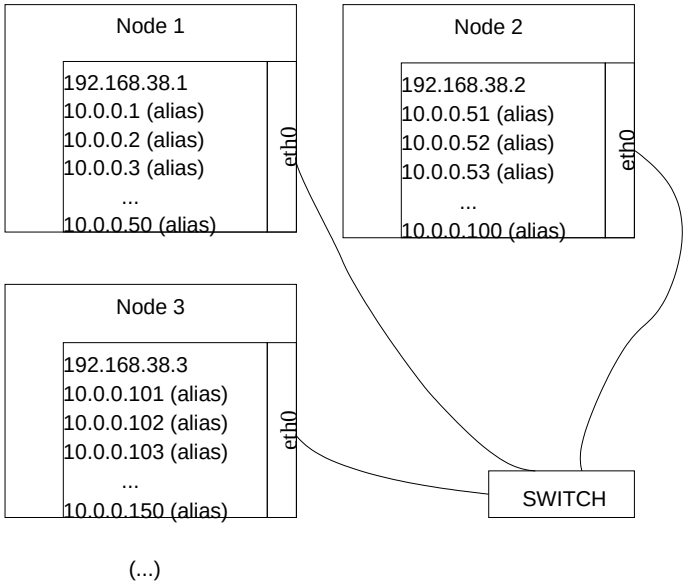

Figure 4. *Sur chaque machine physique, les adresses IP des nœuds virtuels sont configurées comme des alias d'interface*

système d'alias). Nous avons vérifié que les alias d'interface ne provoquaient pas de surcoût par rapport à l'affectation normale d'une adresse IP à une interface.

Pour associer une application à une adresse IP particulière, nous avons choisi d'intercepter les appels systèmes liés au réseau (la figure 5 présente ces différents appels systèmes et leur ordre d'appel). Ainsi, avant chaque appel à `connect()`, un appel à `bind()` est fait pour que l'adresse d'origine de la connexion soit celle souhaitée. De même, les appels à `bind()` sont modifiés pour resteindre le `bind()` à l'adresse souhaitée. Plusieurs solutions étaient possibles pour réaliser techniquement ces restrictions :

– modifier l'application étudiée, par exemple en faisant explicitement un `bind()` avant chaque appel à `connect()`. Cette méthode a le désavantage de nécessiter une compréhension du code de l'application ;

– lier l'application avec une bibliothèque réalisant la surcharge des appels système, soit à la compilation, soit en utilisant la variable d'environnement `LD_PRELOAD`. Cette méthode a le désavantage de ne pas fonctionner si l'application effectue les appels systèmes réseaux à travers une bibliothèque ;

– modifier le noyau pour modifier le traitement des appels systèmes. Sous Linux, c'est la méthode choisie par VServer (Project., 2003). Mais sa mise en œuvre est compliquée ;

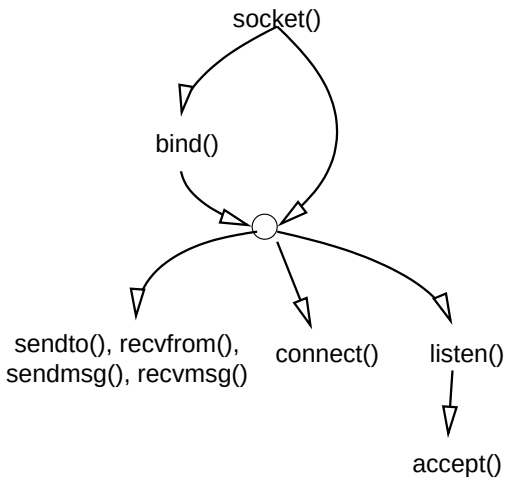

Figure 5. *Enchaînement des appels systèmes réseaux dans le cadre de connexions TCP*

– utiliser `ptrace()` pour intercepter les appels systèmes réseaux. Sous Linux, c'est la méthode choisie par User Mode Linux (Dike, 2000). Cette méthode a un surcoût relativement important à cause des changements de contexte supplémentaires nécessaires ;

– modifier la bibliothèque C. Cette approche permet de se placer « au-dessous » des bibliothèques éventuelles, mais ne fonctionne pas avec les exécutables compilés statiquement.

Dans P2PLab, nous avons choisi de modifier la bibliothèque C de FreeBSD, ce qui est un bon compromis entre complexité et efficacité, et nous avons vérifié que cette approche fonctionnait dans tous les cas, à l'exception des exécutables compilés statiquement. Nous avons modifié les fonctions `bind()`, `connect()` et `listen()` pour toujours se restreindre à l'adresse IP spécifiée dans la variable d'environnement `BINDIP`.

Pour mesurer le surcoût induit par cette technique, nous avons réalisé un programme de test très simple : un client TCP qui se connecte sur un serveur tournant sur la même machine. Dès que la connexion a été acceptée par le serveur, le serveur coupe la connexion. Puis le cycle recommence : le client se reconnecte. En exécutant ce test un nombre important de fois, nous avons pu mesurer avec un intervalle de confiance satisfaisant que la durée d'un cycle était de $10,22\ \mu s$ sans la modification de la `libc`, et de $10,79\ \mu s$ avec la modification de la `libc`. Il faut noter que ce surcoût n'est présent que lors de l'établissement de la connexion. Notre test montre que, même dans un cas limite (application passant tout son temps à établir des connexions), ce surcoût reste très limité comparé au temps nécessaire pour établir une connexion.

Notre méthode de surcharge ne fonctionne que pour les connexions TCP. Une approche identique permettrait dans le futur de traiter également les communications UDP, mais il faudrait alors surcharger les appels `send()` et `receive()`, ce qui provoquerait un surcoût plus élevé : les appels de bibliothèque seraient alors surchargés à chaque envoi de datagramme, et non plus uniquement lors de l'établissement de la connexion.

4.3. *Emulation réseau*

Les émulateurs de topologies réseau existants comme Modelnet (Vahdat *et al.*, 2002) visent une émulation réaliste du cœur du réseau (routage et interactions entre les *systèmes autonomes* (AS) ou entre les principaux routeurs). Mais la plupart des applications pair-à-pair sont exécutées sur des nœuds situés *en bordure d'internet* (Foster *et al.*, 2003), par exemple sur des ordinateurs personnels d'abonnés à l'ADSL. Même si le trafic dans le cœur du réseau peut influencer certains aspects du comportement des systèmes pair-à-pair (la congestion dans le cœur du réseau peut influencer la latence, par exemple), le principal goulot d'étranglement reste le lien entre le système participant et son fournisseur d'accès à internet. Dans le cadre d'une plate-forme d'expérimentation comme P2PLab, on peut donc utiliser un modèle réseau simplifié, et faire abstraction des problèmes de congestion dans le cœur du réseau. Dans tous les cas, des évaluations sur un outil comme Modelnet ou P2PLab ne doivent pas être considérées comme suffisantes, et doivent être complétées par des évaluations sur des systèmes réels comme PlanetLab (Chun *et al.*, 2003) ou DSLLab (DSLLab, 2005), qui permettent de prendre en compte les problèmes de congestion dans le cœur du réseau, ou par des simulations.

Dans P2PLab, nous modélisons donc le réseau internet en mettant l'accent sur le point de vue du nœud participant, en excluant ce qui est moins important de ce point de vue là.

L'émulation réseau est réalisée d'une manière décentralisée : chaque nœud physique se charge de l'émulation réseau pour les nœuds virtuels qu'il héberge, à l'aide de Dummynet (Rizzo, 1997) : il y a ainsi deux règles dans le pare-feu pour chaque nœud virtuel hébergé, l'une pour les paquets entrants à destination de ce nœud virtuel, l'autre pour les paquets sortant en provenance de ce nœud virtuel.

Mais ce modèle permet uniquement d'avoir des paramètres de bande passante et de latence pour chaque nœud virtuel. Il ne permet pas d'exprimer la *proximité* entre deux nœuds. Nous rajoutons donc une notion de groupe de nœud, permettant d'évaluer des applications utilisant la localité. Dans un système réel, ces groupes de nœuds pourraient correspondre à un ensemble de nœuds provenant du même fournisseur d'accès, du même pays, ou du même continent.

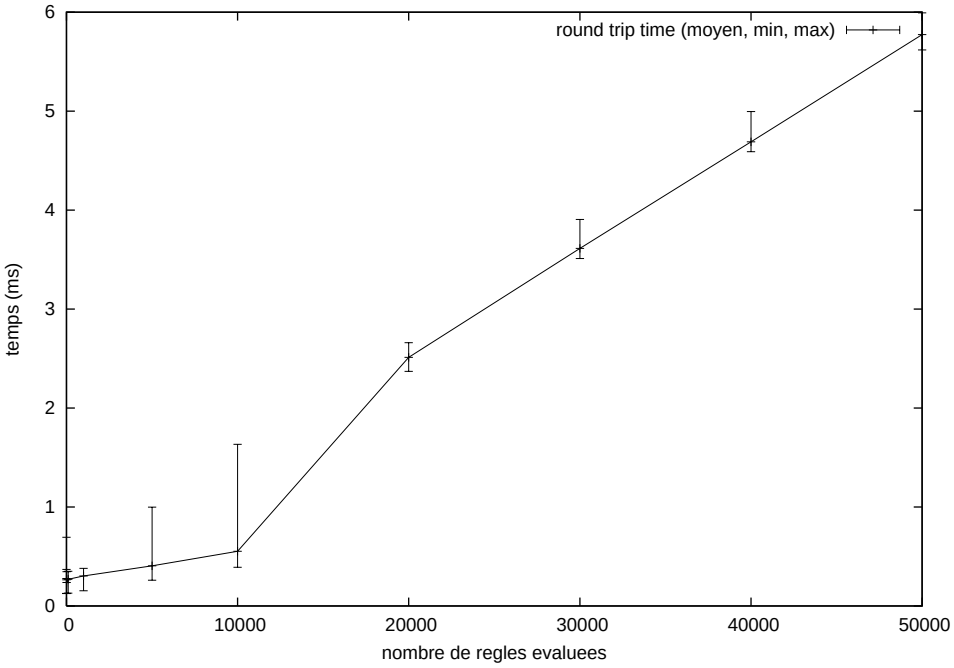

Figure 6. *Evolution du round-trip time en fonction du nombre de règles de pare-feu à évaluer*

Le modèle d'émulation de P2PLab permet de contrôler :

– la bande passante, la latence et le taux de perte de paquets sur les liens réseaux entre les nœuds et leur fournisseur d'accès à internet ;

– la latence entre des groupes de nœuds, permettant d'étudier des problèmes mettant en jeu la localité des nœuds, par exemple.

Le nombre de règles nécessaires est le principal paramètre limitant le passage à l'échelle de P2PLab. Pour montrer l'importance de ce facteur, nous mesurons le *round-trip time* (temps aller/retour) à l'aide de `ping` entre deux nœuds. Lors de la sortie du premier nœud, le pare-feu doit, pour chaque paquet, évaluer un nombre important de règles. La figure 6 permet d'observer que la latence commence par croître assez lentement (probablement à cause d'optimisations), elle croît ensuite linéairement, avec un surcoût non négligeable quand le nombre de règles est important.

Ce surcoût provient du fait que les règles sont évaluées linéairement : avec Dummynet, il n'est pas possible d'évaluer les règles d'une manière hiérarchique, ou à l'aide d'une table de hachage. Une solution possible aurait pû être de modifier le pare-feu de FreeBSD pour y rajouter ce type d'évaluations, mais dans le cadre de P2PLab, nous avons préféré utiliser la notion de groupes de nœuds pour limiter le nombre de règles

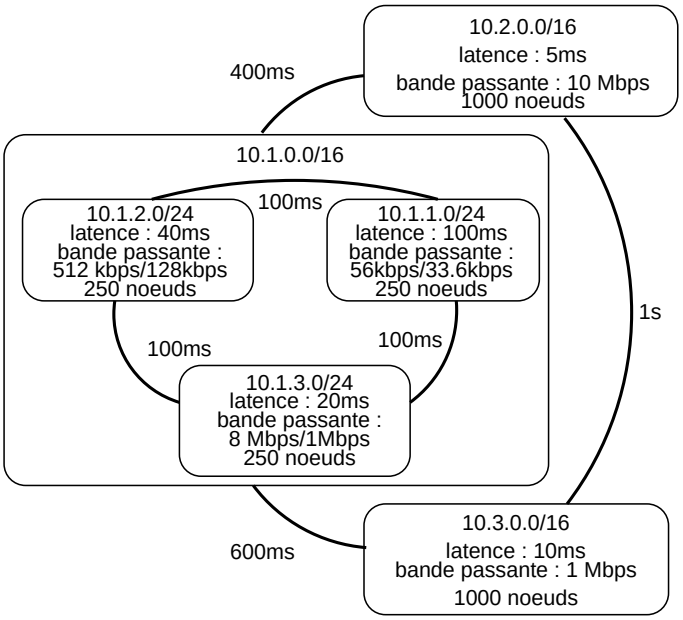

Figure 7. *Topologie émulée*

(une seule règle par groupe permettant alors de traiter la latence pour l'ensemble des nœuds composant le groupe).

La figure 7 montre un exemple de topologie que nous avons émulée avec P2PLab. Le nœud physique hébergeant le nœud virtuel 10.1.3.207 aura par exemple :

– deux règles par nœud physique hébergé (paquets entrants et sortants, respectivement) ;

– une règle pour traiter l'ajout de 100 ms de latence pour les paquets provenant du groupe 10.1.3.0/24 à destination du groupe 10.1.1.0/24 (la règle réciproque étant présente sur les nœuds hébergeant le groupe 10.1.1.0/24) ;

– une règle pour traiter l'ajout de 100 ms de latence pour les paquets provenant du groupe 10.1.3.0/24 à destination du groupe 10.1.2.0/24 ;

– une règle pour traiter l'ajout de 400 ms de latence pour les paquets provenant du groupe 10.1.0.0/16 à destination du groupe 10.2.0.0/16 ;

– une règle pour traiter l'ajout de 600 ms de latence pour les paquets provenant du groupe 10.1.0.0/16 à destination du groupe 10.3.0.0/16.

Nous avons mesuré la latence entre les nœuds 10.1.3.207 et 10.2.2.117 (il n'y avait pas d'autre trafic entre les nœuds). La latence mesurée de 853 ms se décompose en :

– 20 ms de délai lorsque le paquet est parti de 10.1.3.207 (délai pour le groupe 10.1.3.0/24) ;

– 400 ms de délai entre le groupe 10.1.0.0/16 et le groupe 10.2.0.0/16 ;

– 5 ms lorsque le paquet est arrivé sur 10.2.2.117 (délai pour le groupe 10.2.0.0/16) ;

– 425 ms lorsque le paquet est revenu de 10.2.2.117 à 10.1.3.207, comme ci-avant ;

– 3 ms de surcoût, dûs à la latence lors du transit sur le réseau de la grappe, et surtout au parcours des règles du pare-feu.

5. Evaluation d'un système pair-à-pair avec P2PLab : le cas de BitTorrent

Dans cette partie, nous allons réaliser quelques expériences sur BitTorrent (Cohen, 2003) en utilisant P2PLab. Dans un premier temps, nous nous attacherons à montrer l'aptitude de P2PLab à évaluer un tel système pair-à-pair. Puis nous évaluerons certaines techniques permettant à un nœud égoïste d'obtenir de meilleures performances (temps de récupération d'un fichier plus court, ou récupération en consommant moins de bande passante).

BitTorrent est un système pair-à-pair de distribution de fichiers très populaire. Il fournit de très bonnes performances grâce à un mécanisme de réciprocité complexe en s'assurant que les nœuds récupérant un fichier coopèrent en envoyant les morceaux déjà récupérés vers d'autres nœuds. Lors d'un transfert avec BitTorrent, les clients contactent un *tracker* pour récupérer une liste d'autres nœuds participant au téléchargement de ce fichier, puis se connectent directement aux autres nœuds. Les nœuds disposant du fichier complet sont appelés *seeders*.

BitTorrent a déjà été étudié par l'analyse de traces d'une utilisation à grande échelle (Pouwelse *et al.*, 2005; Izal *et al.*, 2004), ainsi qu'à l'aide de modélisation (Qiu *et al.*, 2004) et de simulation (Bharambe *et al.*, 2005). Cependant, ces travaux ont rarement été comparés à des évaluations à grande échelle sur des systèmes réels, ou à des études utilisant l'émulation. BitTorrent est avant tout un travail d'ingénierie, et non un prototype de recherche : plusieurs parties de son code sont très complexes, et le nombre élevé de constantes et de paramètres utilisés dans les algorithmes le rendent très difficile à modéliser avec précision.

Pour toutes les expériences qui suivent, nous avons utilisé les ressources déjà décrites à la section 4.2.1. Les clients BitTorrent utilisés sont BitTorrent 4.4.0 (écrit en Python par l'auteur originel de BitTorrent) et une version modifiée de CTorrent 3.1.4[3] (écrite en C++).

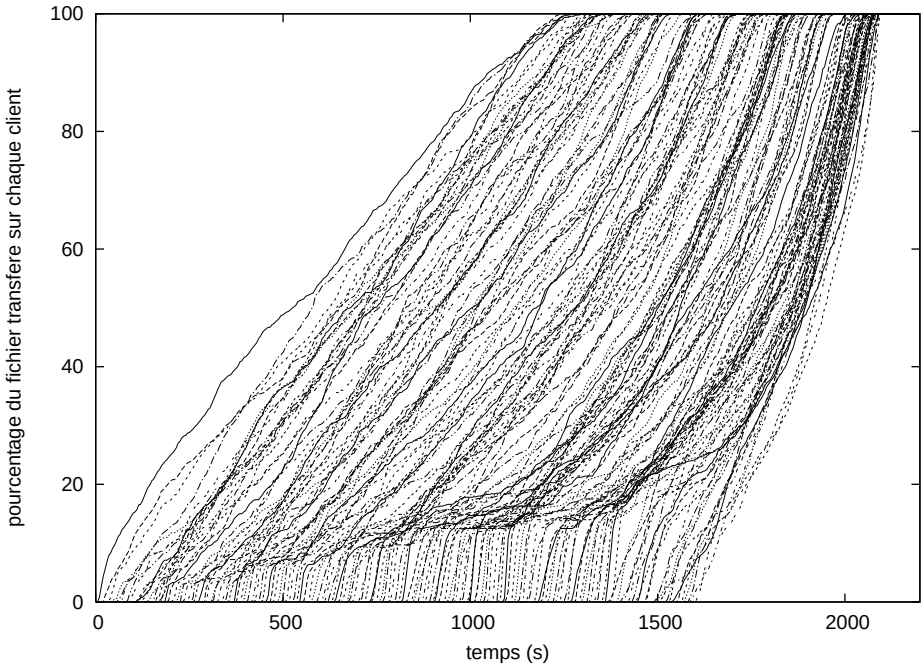

Figure 8. *Evolution du téléchargement des 160 clients*

5.1. *Rapport de virtualisation*

P2PLab vise un bon rapport de virtualisation : il doit être possible d'exécuter de nombreux nœuds virtuels sur le même système physique, permettant ainsi de réaliser des expériences avec un très grand nombre de nœuds. Mais l'exécution concurrente de plusieurs instances de l'application étudiée ne doit pas influencer les résultats.

Dans une première expérience, nous comparons le téléchargement d'un fichier de 16 Mo entre 160 clients. La taille du fichier n'est pas importante dans le cas de BitTorrent, puisque le fichier est toujours découpé en blocs de 256 Ko. Le fichier est fourni par 4 *seeders*. Tous les nœuds (à la fois les téléchargeurs et les *seeders*) disposent des mêmes conditions réseaux :

– bande passante descendante de 2 mbps ;

– bande passante ascendante de 128 kbps ;

– latence (en entrée et en sortie des nœuds) de 30 ms.

Ces conditions sont similaires à celles rencontrées avec des connexions ADSL. Tous les clients ont des caractéristiques identiques : nous avons préféré utiliser ces

3. Disponible sur http://www.rahul.net/dholmes/ctorrent/.

conditions non réalistes, plutôt que de chercher à utiliser des conditions d'expériences supposées réalistes, sans pouvoir le justifier. Quand les clients terminent leur téléchargement, ils restent connectés et transmettent des données aux autres téléchargeurs.

Les 160 clients sont déployés successivements sur 160 nœuds physiques (soit l'ensemble des nœuds disponibles de GridExplorer lorsque cette expérience a été faite), puis 16 nœuds physiques (10 nœuds virtuels par nœud physique), puis 8, puis 4, et 2 nœuds physiques (avec 80 nœuds virtuels par nœud physique).

Les clients sont démarrés toutes les 10 secondes. La figure 8 montre l'évolution du téléchargement sur chacun des 160 clients quand ils sont déployés sur 160 nœuds physiques. On peut constater qu'avec ces conditions expérimentales, toutes les phases d'un téléchargement avec BitTorrent sont représentées :

– d'abord, il y a une période de quelques secondes pendant laquelle seuls les *seeders* sont capables de transmettre des données ;

– puis les téléchargeurs commencent à s'échanger des données ;

– enfin, après environ 1 300 secondes, les premiers téléchargeurs deviennent *seeders* à leur tour, et aident les autres téléchargeurs à finir leur téléchargement.

Ces paramètres sont donc suffisamment réalistes pour vérifier les capacités de virtualisation de P2PLab.

La figure 9 montre que l'expérience s'est réalisée sans surcoût quel que soit le rapport de virtualisation utilisé. Même avec 80 nœuds virtuels par nœud physique, les résultats sont presque identiques (ce qui est amplifié par le fait que nous mesurons la quantité totale de données reçues, ce qui masque les différences éventuelles entre les clients). Les sources potentielles de surcoût ont été recherchées, et il a été déterminé que le premier facteur limitant était la vitesse du réseau : avec des paramètres légèrement différents pour le réseau émulé, le réseau Gigabit de la plate-forme était saturé par les téléchargements.

5.2. *Passage à l'échelle*

Dans cette expérience, nous cherchons à montrer qu'il est possible de réaliser des expériences avec un nombre important de nœuds. Nous transférons un fichier de 16 Mo en utilisant 13 040 nœuds répartis sur 163 machines physiques (80 nœuds virtuels par machine physique). Parmi ces 13 040 nœuds, il y un *tracker*, et 8 nœuds (*seeders*) qui disposent du fichier complet dès le début de l'expérience afin de le transférer vers les autres nœuds. Le *tracker* et les *seeders* sont démarrés au début de l'expérience, tandis que les clients sont démarrés toutes les 0,25 secondes. Les clients (*seeders* compris) à numéro pair utilisent CTorrent, tandis que les clients à numéro impair utilisent BitTorrent 4.4.0. Lorsqu'un client termine le transfert, il reste présent afin d'aider les autres clients.

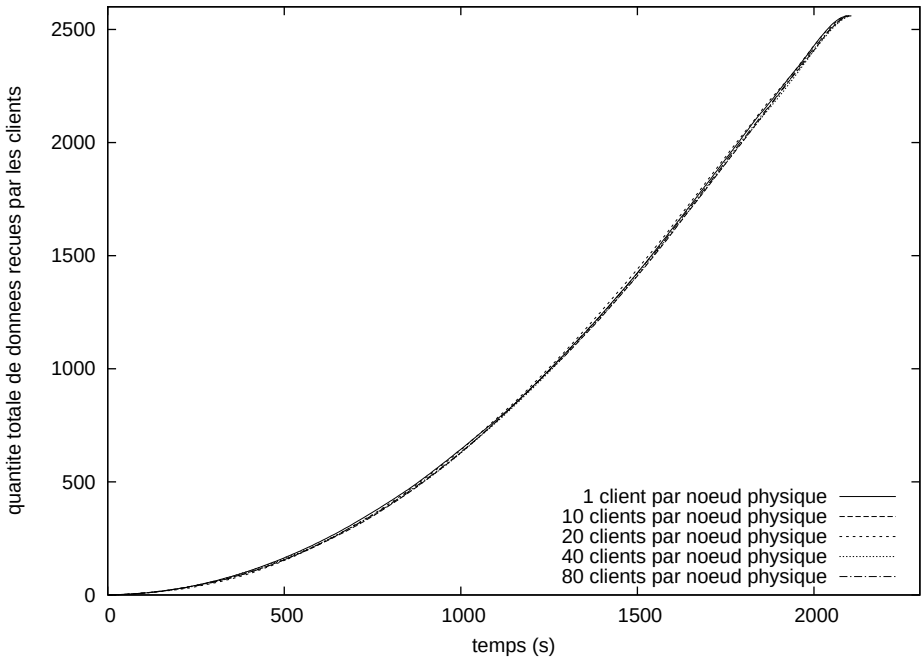

Figure 9. *Rapport de virtualisation de P2PLab : quantité totale de données récupérées par les 160 clients avec les différents déploiements*

La figure 10 montre l'avancement du transfert sur les clients numérotés 101, 200, 301, 400, 501 ... On constate que, malgré des dates de démarrage décalées, la plupart des clients terminent leur transfert à des dates proches, ce qui est confirmé par la figure 11. Cela est dû au mécanisme de réciprocité de BitTorrent : les clients préfèrent aider les clients ayant reçu peu de données que des clients proches de la fin de leur transfert, car ces derniers, une fois le transfert terminé, pourraient quitter le système et ne plus contribuer aux transferts des autres clients.

P2PLab a permis de réaliser cette expérience assez facilement. Toutefois, le nombre important de nœuds virtuels et la fréquence élevée de leurs lancements a causé des problèmes avec les serveurs SSH des nœuds (le serveur SSH n'accepte qu'un nombre faible de connexions en attente, et rejette les connexions en surplus, ce qui pose problème lorsque le rapport de virtualisation est important et que la fréquence de connexion est importante). L'utilisation d'outils spécifiques adaptés au contrôle de ce type d'expériences (lanceurs parallèles) permettrait d'éviter ces problèmes.

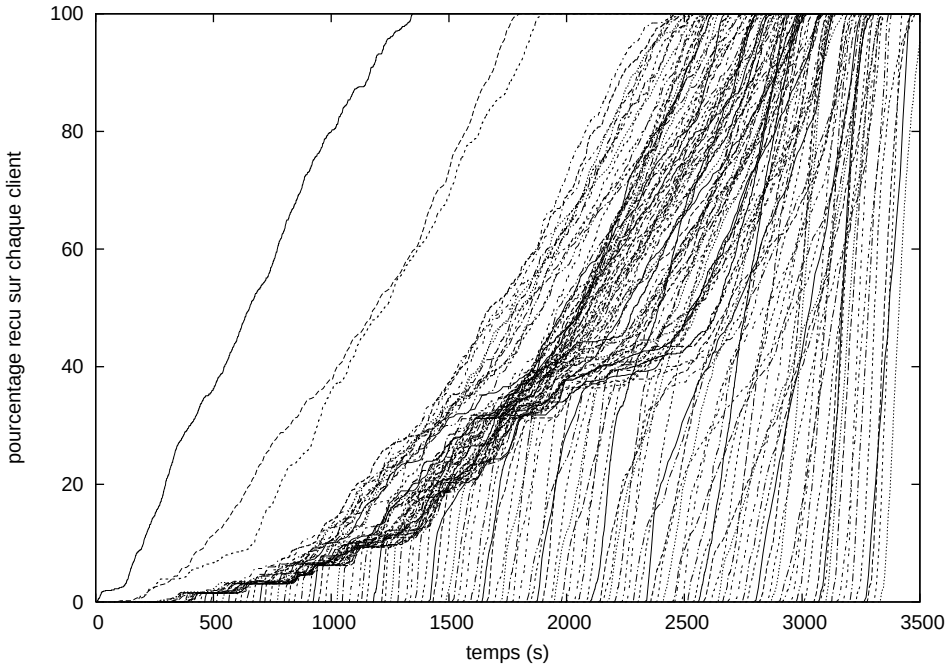

Figure 10. *Evolution du transfert d'un fichier de 16 Mo entre 13 031 clients sur quelques nœuds sélectionnés (nœuds 101, 200, 301, 400 ...)*

5.3. *Comparaison de différentes implantations de BitTorrent*

Puisqu'il permet d'exécuter directement des applications réelles, P2PLab est adapté à la comparaison d'implantations différentes du même protocole. Nous l'utilisons donc pour comparer BitTorrent, le client de référence développé par l'auteur du protocole (Cohen, 2003), et CTorrent, qui a été utilisé dans des travaux visant à montrer certaines limites du protocole (Liogkas *et al.*, 2006). Nous réalisons le transfert d'un fichier de 16 Mo entre 500 clients et 8 *seeders* (4 utilisent BitTorrent, 4 CTorrent) en nous plaçant dans les mêmes conditions réseau que précédemment, et lançons tous les clients en même temps, au début de l'expérience.

Pour vérifier qu'un client BitTorrent est performant, il est important de vérifier qu'il est performant à la fois lorsqu'il ne communique qu'avec des clients identiques, et lorsqu'il communique avec des clients différents. Nous comparons donc les performances de BitTorrent et CTorrent lorsqu'ils sont lancés seuls (500 clients du même type) ou en mélangeant des clients BitTorrent et CTorrent (250 clients de chaque type). Les résultats sont présentés en figures 11 et 12, et montrent que BitTorrent 4.4.0 semble plus performant dans les deux cas avec ces paramètres d'expérience. Il sera toutefois important de compléter cette expérience par d'autres expériences avec

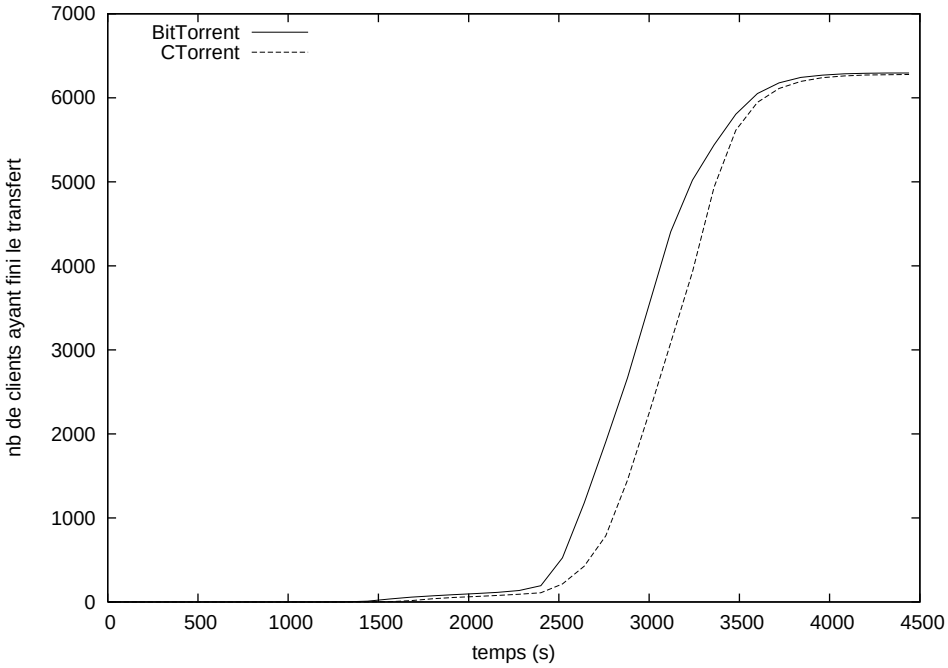

Figure 11. *Temps de complétion du transfert sur les différents nœuds*

des paramètres différents : il est par exemple possible que CTorrent soit plus performant que BitTorrent sous certaines conditions réseau, et notre outil pourrait permettre facilement de comparer ces deux implantations sous un large spectre de conditions réseau.

6. Conclusion

Avec l'augmentation des ressources disponibles sur les ordinateurs, la virtualisation et l'émulation ont fait l'objet de beaucoup d'intérêt ces dernières années. Mais au-delà de leur usage habituel, ils peuvent être combinés pour construire des plates-formes expérimentales permettant une grande configurabilité et la reproductibilité des expériences. Ces plates-formes sont particulièrement utiles dans le cadre de l'étude des systèmes distribués, où le nombre de nœuds participant est un critère primordial.

Dans cet article, nous avons présenté P2PLab, un outil d'étude des systèmes pair-à-pair, et nous avons montré son utilité à travers quelques expériences sur différentes implantations du protocole de diffusion de fichiers BitTorrent.

Nous avons aussi proposé un modèle simple de topologies réseaux virtuelles adapté à l'étude des systèmes pair-à-pair : à la différence des modèles de topologie

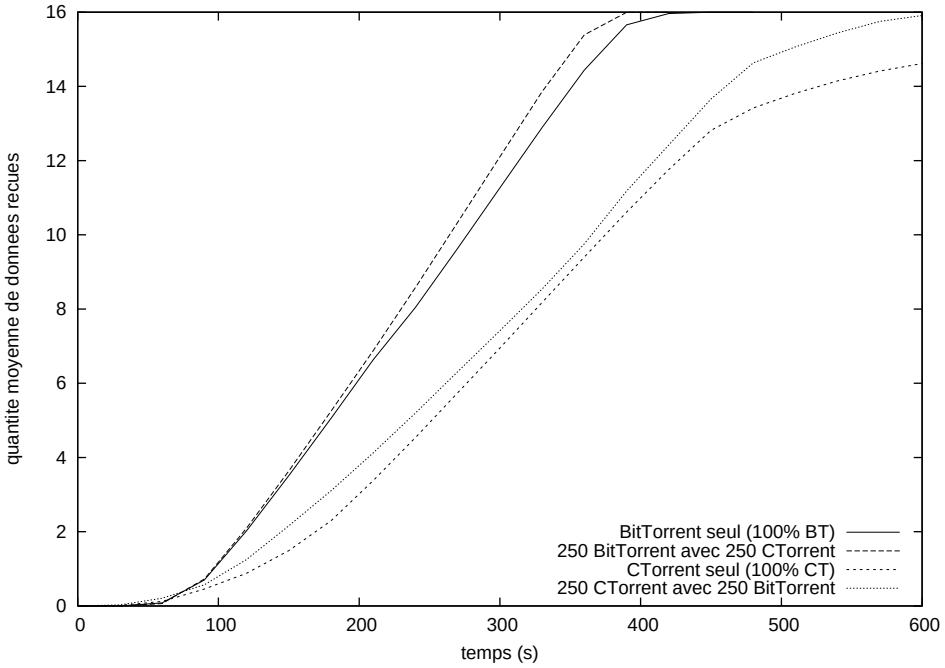

Figure 12. *Comparaison de BitTorrent et CTorrent*

couramment utilisés, nous modélisons le réseau du point de vue du nœud participant, en masquant une partie des interactions se déroulant dans le coeur du réseau. Ce modèle permet aussi une implantation facile dans des outils d'émulation.

Notre travail laisse à penser qu'il n'est pas forcément nécessaire de recourir à des systèmes de virtualisation lourds (virtualisant l'ensemble des ressources). Notre approche plus légère permet d'augmenter le rapport de virtualisation en ne virtualisant que ce qui est absolument nécessaire : l'identité réseau de l'application étudiée.

P2PLab n'est pour l'instant pas disponible publiquement, sa mise en oeuvre nécessitant une expertise trop importante pour qu'il soit facilement utilisable. Les auteurs sont par contre ouverts à toute proposition de collaboration autour de cet outil.

Remerciements

Ce travail a été réalisé au sein du laboratoire ID-IMAG, financé par le CNRS, l'INRIA, l'université Joseph Fournier, l'institut national polytechnique de Grenoble et l'université Pierre Mendès-France. Les différentes évaluations ont été réalisées en

utilisant les ressources des projets GridExplorer et Grid'5000 (plus d'informations sur `http ://www.grid5000.fr/`).

7. Bibliographie

Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., Warfield A., « Xen and the art of virtualization », *SOSP '03 : Proceedings of the nineteenth ACM symposium on Operating systems principles*, ACM Press, New York, NY, USA, 2003.

Bharambe A. R., Herley C., Analyzing and Improving BitTorrent Performance, Technical Report n° MSR-TR-2005-03, Microsoft Research, 2005.

Cappello F., Caron E., Dayde M., Desprez F., Jeannot E., Jegou Y., Lanteri S., Leduc J., Melab N., Mornet G., Namyst R., Primet P., Richard O., « Grid'5000 : a large scale, reconfigurable, controlable and monitorable Grid platform », *Grid'2005 Workshop*, IEEE/ACM, Seattle, USA, November 13-14, 2005.

Carson M., Santay D., « NIST Net : a Linux-based network emulation tool », *SIGCOMM Comput. Commun. Rev.*, vol. 33, n° 3, p. 111-126, 2003.

Chun B., Culler D., Roscoe T., Bavier A., Peterson L., Wawrzoniak M., Bowman M., « PlanetLab : An Overlay Testbed for Broad-Coverage Services », *ACM SIGCOMM Computer Communication Review*, vol. 33, n° 3, p. 3-12, July, 2003.

Cohen B., « Incentives Build Robustness in BitTorrent », 2003, http://www.bittorrent.com.

Dike J., « A user-mode port of the Linux kernel », *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta*, Usenix, p. 63, 2000.

DSLLab, http ://www.lri.fr/~rezmerit/dsllab/, 2005.

Foster I. T., Iamnitchi A., « On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. », *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

Haeberlen A., Mislove A., Post A., Druschel P., « Fallacies in evaluating decentralized systems », *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, February, 2006.

Hemminger S., « Network Emulation with NetEm », *linux.conf.au*, 2005.

Izal M., Urvoy-Keller G., Biersack E. W., Felber P. A., Al Hamra A., Garces-Erice L., « Dissecting BitTorrent : five months in a torrent's lifetime », *PAM'2004, 5th annual Passive & Active Measurement Workshop, April 19-20, 2004, Antibes Juan-les-Pins, France / Also Published in Lecture Notes in Computer Science (LNCS), Volume 3015, Barakat, Chadi ; Pratt, Ian (Eds.) 2004*, Apr, 2004.

Liogkas N., Nelson R., Kohler E., Zhang L., « Exploiting BitTorrent For Fun (But Not Profit) », *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, Santa Barbara, CA, February, 2006.

Nussbaum L., Richard O., « Lightweight Emulation to Study Peer-to-Peer Systems », *Third International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P 06)*, Rhodes Island, Greece, 4, 2006.

Pouwelse J., Garbacki P., Epema D., Sips H., « The Bittorrent P2P File-sharing System : Measurements and Analysis », *4th International Workshop on Peer-to-Peer Systems (IPTPS)*, feb, 2005.

Project. T. L. V., http ://www.linux-vserver.org/Linux-VServer-Paper, 2003.

Qiu D., Srikant R., « Modeling and performance analysis of BitTorrent-like peer-to-peer networks », *SIGCOMM '04 : Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, ACM Press, New York, NY, USA, p. 367-378, 2004.

Rizzo L., « Dummynet : a simple approach to the evaluation of network protocols », *ACM Computer Communication Review*, vol. 27, n° 1, p. 31-41, 1997.

Rizzo L., « Dummynet and Forward Error Correction », *Proceedings of the FreeNIX Track : USENIX 1998 annual technical conference*, Usenix, 1998.

Roberson J., « ULE : A Modern Scheduler for FreeBSD », *Proceedings of BSDCon*, 2003.

Vahdat A., Yocum K., Walsh K., Mahadevan P., Kostic ; D., Chase J., Becker D., « Scalability and accuracy in a large-scale network emulator », *OSDI '02 : Proceedings of the 5th symposium on Operating systems design and implementation*, ACM Press, New York, NY, USA, p. 271-284, 2002.

White B., Lepreau J., Stoller L., Ricci R., Guruprasad S., Newbold M., Hibler M., Barb C., Joglekar A., « An integrated experimental environment for distributed systems and networks », *SIGOPS Oper. Syst. Rev.*, vol. 36, n° SI, p. 255-270, 2002.

Article reçu le 23 janvier 2007
Accepté après révisions le 25 octobre 2007

Lucas Nussbaum *termine son doctorant en informatique à l'université Joseph Fourier de Grenoble. Ses travaux portent sur l'étude des systèmes distribués. Il s'intéresse plus particulièrement à l'utilisation de l'émulation et de la virtualisation dans ce cadre.*

Olivier Richard *est enseignant-chercheur en informatique à l'université Joseph Fourier de Grenoble. Ses travaux portent principalement sur la gestion de resources à grande échelle et les problématiques liées aux plates-formes pour l'expérimentation et à la conduite d'expérience. Il participe notamment à la définition et aux développements de la plate-forme Grid'5000.*